Petre-Florin Stegarus - 323CB

1: bkt

Generez toate combinarile nodurilor de N luate cate k. Pe masura ce obtin o o noua combinare, o si verific daca este k-clica. Daca da, afisez True si opresc programul. Daca nu, generez urmatoarea combinare. Tot asa pana am epuizat toate combinarile, caz in care inseamna ca nu s-a gasit nicio k-clica deci afisez False.
Complexitatea este O(n^k * k^2)

2: rdc

La laboratorul 8, ni s-a prezentat reducerea SAT <=p k-Clique, pe care incerc sa o inversez. Construiesc un grafic asemanator si in cazul k-Clique <=p SAT ca in cazul de la laborator, in functie de care construiesc clauzele.

```
(
  Exemplul din laborator:
  SAT <=p k-Clique: (phi = c1 ^ c2 ^ ... ^ cn)
  k = n
  G:
    V = {xvi pt fiecare Lvi din fiecare ci}
    E = (xui, xvj) apartine E daca i != j, Lui != ~Lvj
)
```

k-Clique(k, G) <=p SAT(phi):

(Lvi = True daca nodul v este al i-lea nod din k-clica)
n = k

(1) Pentru fiecare i, unul dintre noduri trebuie sa fie al i-lea nod din clica. Deci am cate o clauza ci (i=1..k) cu literalii Lvi pentru fiecare nod v=1..N (ci = L1i V L2i V ... V LNi).

(2) Totodata, acelasi nod nu poate aparea de mai multe ori in clica. Deci, pentru oricare nod v, (~Lvi V ~Lvj) oricare i != j

(3) Daca intre 2 noduri nu este muchie, nu pot fi amandoua intr-o clica. Deci dacă u si v nu sunt adiacente, am clauza (~Lui V ~Lvj) pentru orice i != j (i, j fiind perechi de "nivele" din clica, 1<=i<j<=k)

inputul pentru SAT va fi phi = conjunctia tuturor clauzelor obtinute din (1), (2), (3)

(1) k clauze cu cate N literali: N * k
(2) (N * combinari de k luate cate 2) clauze cu 2 literali: N * k * (k - 1)
(3) combinari de N luate cate 2 perechi de muchii * combinari de k luate cate 2 perechi (i, j) clauze cu 2 literali: N * (N - 1) * k * (k - 1)
Adunand (1), (2), (3) obtin N * k * (N * k - N + 1) adica O(N^2 * k^2) clauze.
Cum k < N si k depinde de N, complexitatea transformarii este O(N^4), deci polinomiala.

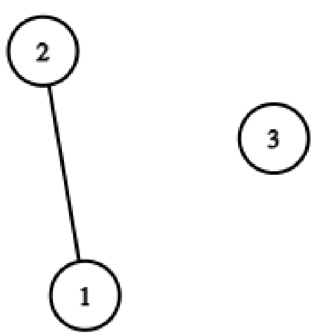
Demonstratie k-Clique(k, G) = 1 <=> SAT(phi) = 1

1) k-Clique(k, G) = 1 => SAT(phi) = 1
k-Clique(k, G) = 1 <=> Exista k noduri distincte adiacente 2 cate 2 care sa formeze o clica, [v1, v2,..., vk] (vi fiind din lista de noduri si distincte 2 cate 2).
Deci exista literalii Lv1,1 Lv2,2 ... Lvk,k care sunt adevarati si astfel sunt indeplinite cele 3 tipuri de clauze:
- Exista k noduri in clica -> indeplineste clauzele de tipul (1)
- Faptul ca sunt noduri distincte indeplineste (2)
- Adiacente 2 cate 2 indeplineste (3)
De unde rezulta ca SAT(phi) = 1

2) SAT(phi) = 1 => k-Clique(k, G) = 1
SAT(phi) = 1 <=> oricare clauza ci este adevarata => exista literalii Lv1,1 Lv2,2 ... Lvk,k care sunt adevarati (din clauzele care impun cele 3 conditii: exista k astfel de literali, vizeaza noduri distincte, nodurile vizate au toate muchii intre ele) => nodurile [v1, v2,..., vk] (unde vi distincte 2 cate 2, adiacente 2 cate 2) formeaza o k-clica => k-Clique(k, G) = 1

Exemple:

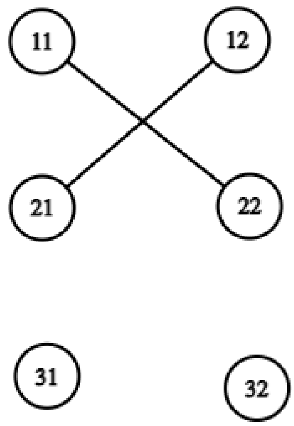
Exemplu 1:
Fie k = 2 si graful G cu 3 noduri si muchia (1, 2).



Imi imaginez graful cu k * N = 6 noduri $x_{v,i}$, pentru fiecare nod v al grafului original si pentru i = 1..k.

Petre-Florin Stegarus - 323CB

Deci, ca exemplu:
11 e pentru nodul 1, pozitia 1 in clica
21 e pentru nodul 2, pozitia 1 in clica
12 e pentru nodul 1, pozitia 2 in clica



Am asezat pe coloane nodurile grafului initial, coloana c_i avand al i-lea element din clica.

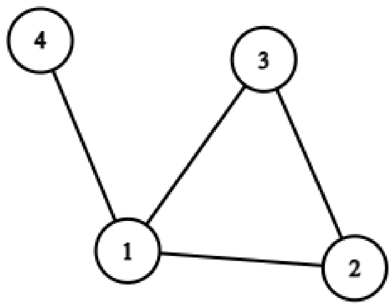
Din graful rezultat, construiesc clauzele, conform transformarii polinomiale (O(N^4)) din README:

(1) (11 V 21 V 31)
(12 V 22 V 32)
(2) (~11 V ~12)
(~21 V ~22)
(~31 V ~32)
(3) (~11 V ~32)
(~21 V ~32)
(~31 V ~12)
(~31 V ~22)

Phi = (11 V 21 V 31) ^ (12 V 22 V 32) ^ (~11 V ~12) ^ (~21 V ~22) ^
^ (~31 V ~32) ^ (~11 V ~32) ^ (~21 V ~32) ^ (~31 V ~12) ^ (~31 V ~22)
Phi este adevarata doar pentru perechile (11 True, 22 True, restul False) si (21 True, 12 True, restul False) => 2-clica formata din nodurile (1, 2)

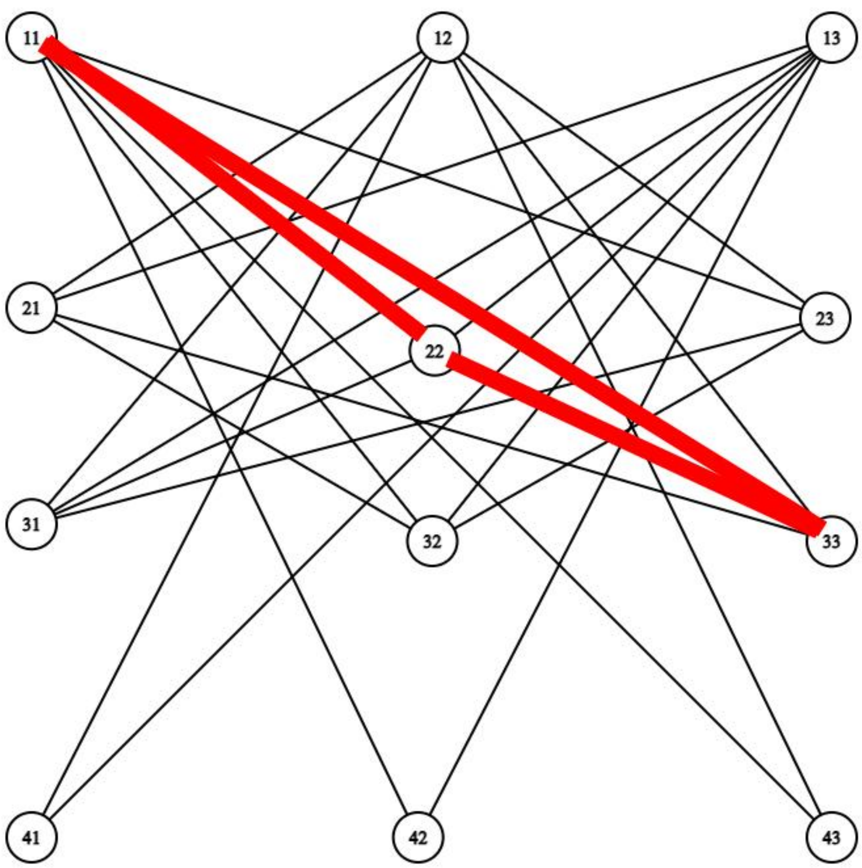
Petre-Florin Stegarus - 323CB

Exemplu 2:



Fie k = 3 si graful G cu 4 noduri si muchiile
1 2
1 3
2 3
1 4

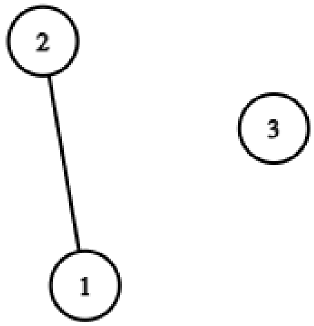
Imi imaginez graful cu k * N = 12 noduri $x_{v,i}$, pentru fiecare nod v al grafului original si pentru i = 1..k.



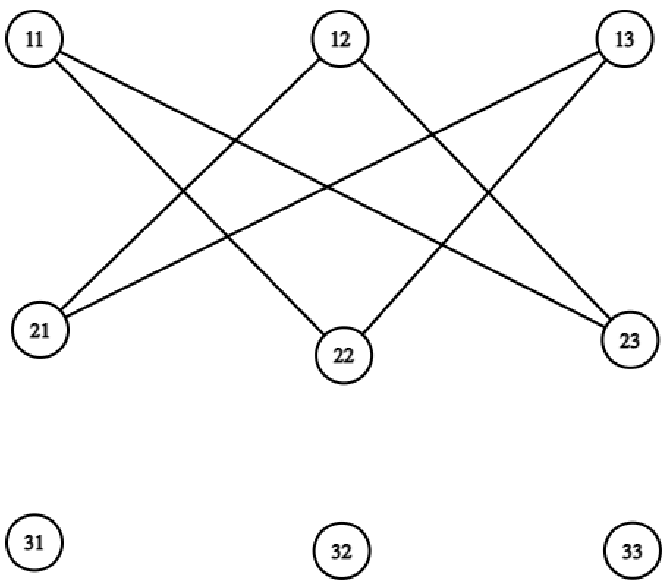
Clauzele se obtin ca mai sus. Am evidentiat cu rosu una din perechile care vor fi gasite rezolvand formula obtinuta cu un SAT solver.

Petre-Florin Stegarus - 323CB

Exemplu 3:
Fie k = 3 si graful G cu 3 noduri si muchia (1, 2).



Imi imaginez graful cu k * N = 9 noduri $x_{v,i}$, pentru fiecare nod v al grafului original si pentru i = 1..k.
Se observa ca nu exista nicio 3-clica



Din graful rezultat, construiesc clauzele, conform transformarii polinomiale (O(N^4)) din README:

(1) (11 V 21 V 31)
(12 V 22 V 32)
(13 V 23 V 33)

(2) (~11 V ~12)
(~11 V ~13)
(~12 V ~13)
(~21 V ~22)
(~21 V ~23)

(~22 V ~23)
(~31 V ~32)
(~31 V ~33)
(~32 V ~33)
(3) (~11 V ~32)
(~11 V ~33)
(~12 V ~33)
(~21 V ~32)
(~21 V ~33)
(~22 V ~33)
(~31 V ~12)
(~31 V ~22)
(~32 V ~13)
(~32 V ~23)

Phi = conjunctia clauzelor de mai sus. Nu are solutie => nu exista 3-clica, ceea ce este adevarat.

Rezultate, comparatia implementarilor si cazuri & exemple favorabile:

Se pare ca in toate categoriile primite, backtrackingul dureaza mai putin decat reducerea. Am facut totusi la bkt optimizarea ca nu caut subgrafuri cu dimensiunea mai mare decat k. Si transformarea pentru rdc este cu siguranta polinomiala.

Cred ca faptul asta se datoreaza unui k prea mic sau prea mare, caz in care backtrackingul ruleaza intr-un timp chiar polinomial.
De exemplu, pentru un N mare (N > 100) si un k mic (ex k = 3), complexitatea este O(N^3) pentru bkt. Dar, daca ar fi sa aleg teste in care aleg k ca fiind N/2 (gen N=100, k=50), se observa ca bkt dureaza mult prea mult (O(N^N) pentru ca k depinde de N).
Local, pentru N=50 si k=25, bkt ruleaza de cateva minute si inca nu s-a terminat. Pana si N=50 si k=6 e prea mult pentru bkt, cu tot cu optimizarile, in timp ce reducerea rezolva relativ repede. (N=50, k=6: bkt=92.8s, rdc=6.7s) Pe de alta parte, pentru k=N, bkt este instant in timp ce rdc dureaza prea mult (n=50, k=50: bkt=0.06s, rdc=384.7s).

De aici deduc ca rdc este mai optim pentru un k din zona mai din mijloc a lui range(1,N) (k=N/2 fiind cel mai favorabil pentru rdc), in timp ce btk este mai optim pentru k din extremitatile range-ului (k=1 sau k=N cel mai favorabil pentru bkt, dar pentru k=N se obtine cea mai mare diferenta intre rdc si bkt in favoarea bkt)

Deci, pentru un k suficient de apropiat de N/2, este mult mai optima rezolvarea cu reducerea. Deci k = N/2 ar face cea mai mare diferenta in favoarea reducerii. Totusi, pentru un numar mic de noduri N, nu se poate observa acest lucru. Din acest motiv am mai facut o categorie, category4, in care am pus 2 teste care sunt in favoarea reducerii. Le-am atasat in caz ca vreti sa le incercati (local: bkt=12.7s rdc=5.1s)

Pe scurt, pentru N suficient de mare si k suficient de apropiat de N/2, bkt dureaza mult prea mult, in timp ce rdc este foarte rapid. Apoi, cu cat creste k si se departeaza de N/2, bkt incepe sa devina mai eficient decat rdc.